NASA-IVV-97-016

NASA IV&V Facility, Fairmont, West Virginia

A Framework for Performing Verification and Validation in Reuse Based Software Engineering

Edward A. Addy

November 11, 1997

This technical report is a product of the National Aeronautics and Space Administration (NASA) Software Program, an agency wide program to promote continual improvement of software engineering within NASA. The goals and strategies of this program are documented in the NASA software strategic plan, July 13, 1995.

Additional information is available from the NASA Software IV&V Facility on the World Wide Web site http://www.ivv.nasa.gov/

This research was funded under cooperative Agreement #NCC 2-979 at the NASA/WVU Software Research Laboratory.

Article to be published in the Annals of Software Engineering,
Special Volume on Reuse, 1998

A Framework for Performing Verification and Validation in Reuse-Based Software Engineering

Edward A. Addy

NASA/WVU Software Research Laboratory
NASA/WVU Software IV&V Facility
100 University Drive
Fairmont, WV 26554 USA
eaddy@wvu.edu

Verification and Validation (V&V) is currently performed during application development for many systems, especially safety-critical and mission-critical systems. The V&V process is intended to discover errors, especially errors related to critical processing, as early as possible during the development process. The system application provides the context under which the software artifacts are validated.

This paper describes a framework that extends V&V from an individual application system to a product line of systems that are developed within an architecture-based software engineering environment. This framework includes the activities of traditional application-level V&V, and extends these activities into domain engineering and into the transition between domain engineering and application engineering. The framework includes descriptions of the types of activities to be performed during each of the life-cycle phases, and provides motivation for the activities.

1. INTRODUCTION

The implementation of reuse-based software engineering not only introduces new activities to the software development process, such as domain analysis and domain modeling, it also impacts other aspects of software engineering. Other areas of software engineering that are affected include Configuration Management, Testing, Quality Control, and Verification and Validation (V&V). Activities in each of these areas must be adapted to address the entire domain or product line rather than a specific application. This paper discusses changes and enhancements to V&V methods that provide a framework for performing V&V within reuse-based software engineering.

V&V methods are used to increase the level of assurance of critical software, particularly that of safety-critical and mission-critical software. Software V&V is a systems engineering discipline that evaluates software in a systems context [Wallace and Fujii 1989a]. The V&V methodology has been used in concert with various software development paradigms, but always in the context of developing a specific application system. However, the reuse-based software development process separates domain engineering from application engineering in order to develop generic reusable software components that are appropriate for use in multiple applications.

The Glossary of Software Reuse Terms published by the National Institute of Standards and Technology [Katz et al. 1994] defines a domain as a distinct functional area that can be supported by a class of software systems with similar requirements and capabilities. Domain analysis is defined as the process by which information used in developing software systems is identified, captured, and organized so that it can be reused to create new systems within a domain. Domain analysis results in a model of the domain, and ultimately in a domain architecture. If the domain corresponds to a line of products, which it often does, the domain architecture is used to guide the development of repeated application systems within the domain. This view is consistent with that of the Defense Advanced Research Project Agency Domain-Specific Software Architecture program [Armitage 1993].

The earlier a problem is discovered in the development process, the less costly it is to correct the problem. To take advantage of this, V&V begins verification within system application development at the concept or high-level requirements phase. However, a reuse-based software development process has tasks that are performed earlier, and possibly much earlier, than high-level requirements for a particular application system.

In order to bring the effectiveness of V&V to bear within a reuse-based software development process, V&V must be incorporated within the domain engineering process. Failure to incorporate V&V within domain engineering will result in higher development and maintenance costs due to losing the opportunity to discover problems in early stages of development and having to correct problems in multiple systems already in operation. Also, the same V&V activities will have to be performed for each application system having mission or safety-critical functions.

On the other hand, it is not possible for all V&V activities to be transferred into domain engineering, since verification extends to the installation and operation phases of development and validation is primarily performed using a developed system. This leads to the question of which existing (and/or new) V&V activities would be more effectively performed in domain engineering rather than in (or in addition to) application engineering. Related questions include how to identify the reusable components for which V&V at the domain level would be cost-effective, and how to determine the level to which V&V should be performed on the reusable components.

This paper describes a framework for performing V&V within reuse-based software engineering. The framework identifies V&V tasks that could be performed in domain engineering, V&V tasks that could be performed in the transition from domain engineering to application engineering, and the impact of these tasks on application V&V activities. The criteria and motivation for performing V&V in domain engineering are also considered.

2. VERIFICATION AND VALIDATION IN TRADITIONAL SYSTEM APPLICATION ENGINEERING

V&V has been performed during application system development, within the context of many different development methodologies, including waterfall, spiral, and evolutionary development. V&V is a set of activities performed in parallel with system development and designed to provide assurance that a software system meets the operational needs of the user. It ensures that the requirements for the system are correct, complete, and consistent, and that the life-cycle products correctly implement system requirements.

The term *verification* refers to the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase, while *validation* is the process of evaluating software at the end of the software development process to ensure compliance with software requirements [IEEE Std 610.12-1990]. Verification is intended to ensure that the product is built correctly, while validation assures that the correct product is built.

While verification and validation have separate definitions, in practice the activities are merged into a single process. This process evaluates software in a systems context, using a structured approach to analyze and test the software against system functions and against hardware, user and other software interfaces [Wallace and Fujii 1989a]. V&V is also described as a series of technical and management activities performed to improve the quality and reliability of that system and to assure that the delivered product satisfies the user's operational needs [Lewis 1992].

V&V activities are designed to be independent of but complementary to the activities of the development and test teams. Where the development team is usually focused on nominal performance and the testing is usually based on requirements and operational profiles, V&V includes analysis and tests on critical and off-nominal behavior throughout all phases of the development lifecycle. V&V activities also complement the activities of the configuration management and quality assurance groups rather than being a duplicate or replacement of these activities [Wallace and Fujii 1989b].

A set of minimal and optional V&V activities is defined in the IEEE Standard for Software Verification and Validation Plans [1986 (R 1992)]. The minimum V&V tasks for critical software are shown in Figure 1.

- Management of V&V
- Concept Phase V&V
- Requirements Phase V&V
- Design Phase V&V
- Implementation Phase V&V
- Test Phase V&V
- Installation and Checkout Phase V&V
- Operations and Maintenance Phase V&V

V&V is performed as a part of a risk mitigation strategy for application systems. The risks can be in areas such as safety, security, mission, finance, or reputation. The scope and level of V&V can vary with each project, based on the criticality of the system and on the role of software in accomplishing critical functions of the system [Makowsky 1992]. V&V determines the software involved in high-risk areas, and V&V activities are focused on this critical software. Criticality analysis is used to determine not only the critical software, but also the level of intensity to which each V&V task should be performed on various portions of the critical software [IEEE Std 1059-1993].

3. DIFFERENCES BETWEEN V&V AND COMPONENT CERTIFICATION

Much work has been done in the area of component certification, which is also called evaluation, assessment, or qualification. These terms can have slightly different meanings, but refer in general to rating a reusable component against a specified set of criteria.

Reuse libraries often use levels to indicate the degree to which a component has been evaluated by the library. The Asset Source for Software Engineering Technology (ASSET) library and the Army Reuse Center library both have four levels of certification, although the use of the term "levels" is operationally different in the two libraries [Poore et al. 1992]. Component-based libraries evaluate reusable components against criteria such as reusability, evolvability, maintainability, and portability, as well as expending various levels of effort to ensure the component meets its specification.

The Certification of Reusable Software Components Program at Rome Laboratory has proposed a certification framework based on removing defects from candidate reusable

PHASE	TASKS
Management	Software Verification and Validation Plan Generation Baseline Change Assessment Management Review Review Support
Concept	Concept Documentation Review
Requirements	Software Requirements Traceability Analysis Software Requirements Evaluation Software Requirements Interface Analysis System Test Plan Generation Acceptance Test Plan Generation
Design	Design Traceability Analysis Design Evaluation Design Interface Analysis Component Test Plan Generation Integration Test Plan Generation Test Design Generation • component testing • integration testing • system testing • acceptance testing
Implementation	Source Code Traceability Analysis Source Code Evaluation Source Code Interface Analysis Source Code Documentation Evaluation Test Case Generation • component testing • integration testing • system testing • acceptance testing Test Procedure Generation • component testing • integration testing • system testing Component Test Execution
Test	Test Procedure Generation • acceptance testing Integration Test Execution System Test Execution Acceptance Test Execution
Installation and Checkout	Installation Configuration Audit V&V Final Report Generation
Operations and Maintenance	Software V&V Plan Revision Anomaly Evaluation Proposed Change Assessment Phase Task Iteration

Figure 1: Minimum V&V Tasks for Critical Software in Application Engineering

components [Software Productivity Solutions 1996]. This certification process consists of four levels of analysis and testing, each designed to remove certain categories of defects from the reusable component. The levels of analysis and testing correspond to more stringent levels of certification, which are composed of the factors of scope and confidence.

The Comprehensive Approach to Reusable Defense Software (CARDS) library is a model-based library based on a generic architecture. Reusable components are evaluated not only on the same general criteria as that of component-based libraries, but also on the "form, fit, and function" relative to the generic architecture [Unisys and EWA 1994]. The CARDS library uses this difference to draw a distinction between "certification" and "qualification". The Component Providers and Tool Developers Handbook defines component certification as "The process of determining if a component being considered for inclusion in a library meets the requirements of the library and passes all testing procedures. Evaluation takes place against a common set of criteria (reusability, portability, etc.)." Component qualification is defined as "The process of determining if a potential component is appropriate to the library and meets all quality requirements. Evaluation takes place against domain criteria."

The common thread through all of these certification processes is the focus on the component rather than on the systems in which the component will eventually be (re)used. Dunn and Knight [1993] note that with the exception of the software industry itself, customers purchase systems and not components. Ensuring that components are well designed and reliable with respect to their specifications is necessary but not sufficient to show that the final system meets the needs of the user. Component evaluation is but one part of an overall V&V effort, analogous to code evaluation in V&V of an application system.

Another distinction between V&V and component certification is the scope of the artifacts that are considered. While component certification is primarily focused on the evaluation of reusable components (usually code-level components), V&V also considers the domain model and the generic architecture, along with the connections between domain artifacts and application system artifacts. Some level of component certification should be performed for all reusable components, but V&V is not always appropriate. V&V should be conducted at the level determined by an overall risk mitigation strategy.

4. JUSTIFICATION FOR PERFORMING V&V WITHIN DOMAIN ENGINEERING

Studies have shown that the cost and difficulty of correcting an error increases dramatically as the error is discovered in later life-cycle phases [Makowsky 1992]. V&V addresses that issue in traditional system development through activities that begin in the concept or high-level requirements phase and continue throughout all life-cycle phases. The V&V activities are focused on high-risk areas, so that errors in the high-risk areas can be discovered in time to evolve a complete and cost effective solution rather than forcing a makeshift solution due to schedule constraints.

Within reuse-based software engineering, software engineering activities may be performed prior to the concept phase of a particular application system. In order to extend the benefit of early error detection to reuse-based software engineering, V&V must be incorporated within the domain engineering process. Performing V&V at the domain level may also reduce the level of effort required to perform V&V in the individual application systems.

Although software is the target of V&V activities, V&V recognizes that software does not execute in isolation, but is an integral part of a system [Duke 1989]. In order to provide assurance that critical functions will be performed correctly, software must be evaluated within the context in which the software will execute. In reuse-based software engineering, the context for V&V must be provided by the domain model and domain architecture.

5. FRAMEWORK FOR PERFORMING V&V WITHIN REUSE-BASED SOFTWARE ENGINEERING

One model for reuse-based software engineering is the Two Life-Cycle Model shown in Figure 2, developed by the U.S. Department of Defense Software for Adaptable, Reliable Systems (STARS) program. This model assumes a domain-specific, architecture-centered approach to software reuse. The domain model describes the problem space of the domain, and expresses requirements. The domain architecture describes the solution space of the domain, while the domain components are intended to be used within application systems to meet the functions described in the domain architecture.

A draft framework for performing V&V within reuse-based software engineering is formed by adding V&V activities to the STARS Two Life-Cycle Model. The application-level IV&V tasks described in IEEE STD 1012 serve as a starting point. Domain-level tasks are added to link life-cycle phases in the domain level, and transition tasks are added to link application phases with domain phases. This draft framework was refined by a working group at Reuse '96 [Addy 1996], and the resultant framework is shown in Figure 3. The specific tasks of each phase at the domain and transition levels are listed in Figure 4.

Domain-level V&V tasks are performed to ensure that domain products fulfill the requirements established during earlier phases of domain engineering. Transition- level tasks provide assurance that an application artifact correctly implements the corresponding domain artifact. Traditional application-level V&V tasks ensure the application products fulfill the requirements established during previous application life-cycle phases.

Performing V&V tasks at the domain and transition levels will not automatically eliminate any V&V tasks at the application level. However, it might be possible to reduce the level of effort for some application-level tasks. The reduction in effort could occur in a case where the application artifact is used in an unmodified form from the domain component, or where the application artifact is an instantiation of the domain component through parameter resolution or through generation.

Domain maintenance and evolution are handled in a manner similar to that described in the operations and maintenance phase of application-level V&V. Changes proposed to domain artifacts are assessed by V&V to determine the impact of the proposed correction or enhancement. If the assessment determines that the change will impact a critical area or function within the domain, appropriate V&V activities are repeated to assure the correct implementation of the change.

Although not shown as a specific V&V task for any particular phase of the life-cycle, criticality analysis is an integral part of V&V planning. Criticality analysis is performed in V&V of application development in order to allocate V&V resources to the most important (i.e., critical) areas of the software [IEEE Std 1059-1993]. This assessment of criticality and the ensuing determination of the level of intensity for V&V tasks are crucial also within reuse-based

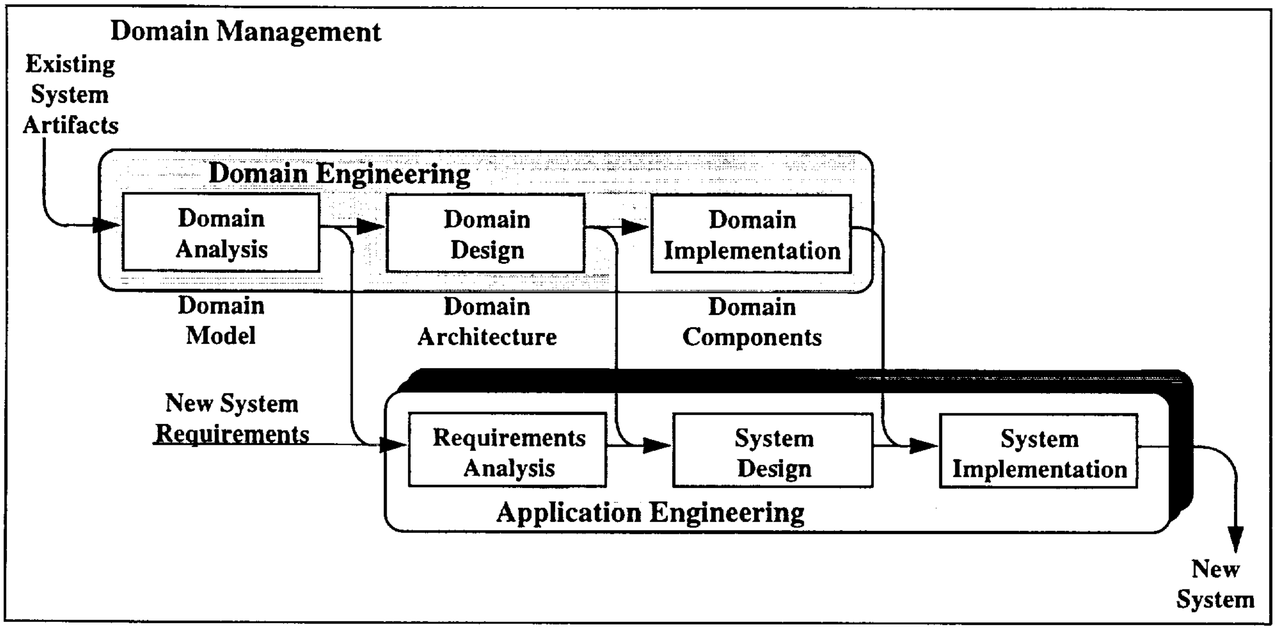

Figure 2: STARS Two Life-Cycle Model

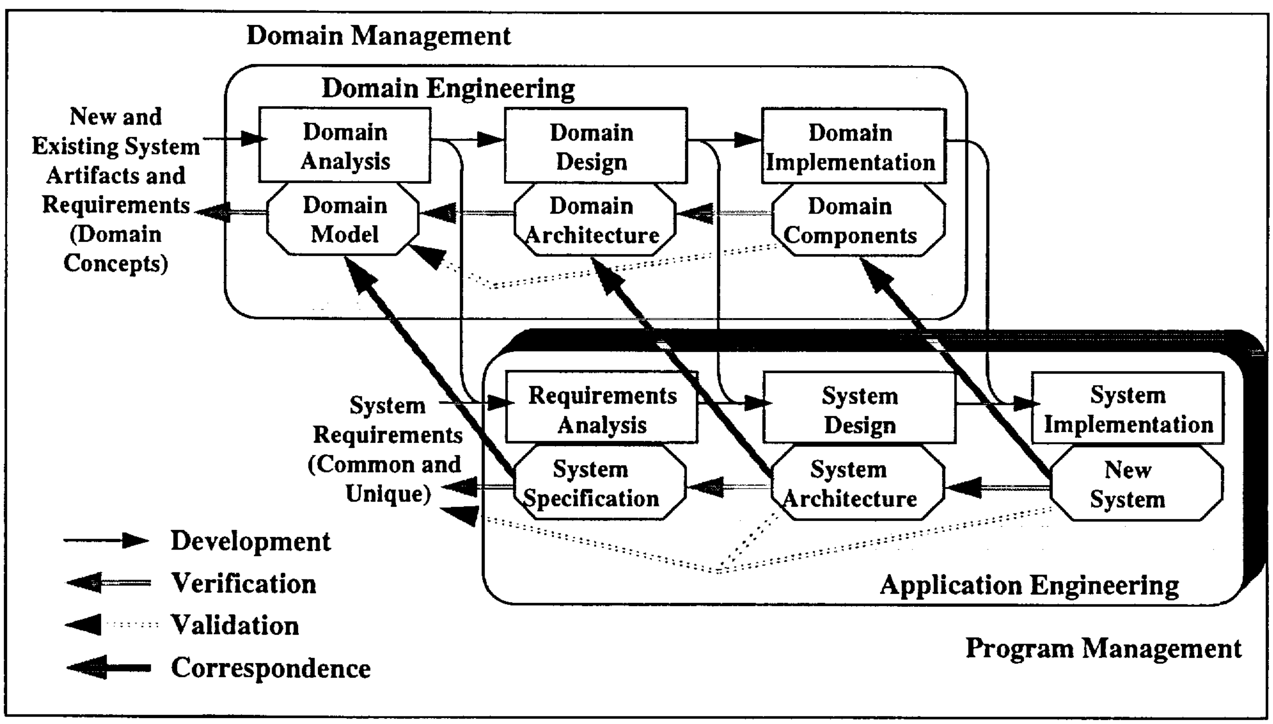

Figure 3: Framework for V&V within Reuse-Based Software Engineering

LEVEL	PHASE	TASKS
Domain Engineering	Domain Analysis	Validate Domain Model Model Evaluation Requirements Traceability Analysis (especially forward traceability for completeness)
	Domain Design	Verify Domain Architecture Design Traceability Analysis Design Evaluation Design Interface Analysis Component Test Plan Generation Component Test Design Generation
	Domain Implementation	Verify and Validate Domain Components Component Traceability Analysis Component Evaluation Component Interface Analysis Component Documentation Evaluation Component Test Case Generation Component Test Procedure Generation Component Test Execution
Transition	Requirements	Correspondence Analysis between System Specification and Domain Model
	Design	Correspondence Analysis between System Architecture and Domain Architecture
	Implementation	Correspondence Analysis between System Implementation and Domain Components

Figure 4: V&V Tasks for Life-Cycle Phases at the Domain and Transition Levels

software engineering. Not all domain products will be used in critical application systems, and some of those used in critical application systems may not be in a critical area of the software. Some reusable components may be used in multiple systems, but may be a part of the critical software in only one or two of the systems. V&V should be performed only on domain products that are involved in the critical software in one or more application systems, and V&V tasks should be performed at a level of intensity appropriate to the level of criticality. Determining the domain products for which to perform V&V, and the appropriate level of intensity for the V&V tasks, is complicated by the use of the products in multiple systems, some of which may only be in early stages of planning. If a component is used in only one critical application system, it may be more cost-effective to perform V&V during application engineering for that system rather than during domain engineering. Extension of criticality analysis from application engineering to domain engineering is an important, but not yet well-defined, area of this framework.

5.1 Domain-Level Tasks

The domain-level tasks are analogous to the application-level tasks, in that the products of each phase are evaluated against the requirements specified in the previous stage and against the original user requirements. The domain-level tasks can be divided into the three phases of domain analysis, domain design, and domain implementation, which correspond to the application phases of requirements, design, and implementation.

During domain analysis V&V, the V&V team should ensure that the domain model is an appropriate representation of the user requirements. (The singular term "model" is not intended to imply that only one model will be constructed; this term is used to mean the one or more models that express the domain requirements.) Note that ensuring that user requirements are satisfied implies that the requirements in the domain must be explicitly stated. Criticality analysis is performed to ensure that high risk requirements are appropriately addressed, either mission-critical requirements or those related to properties such as safety and security. The criticality analysis should also determine critical functions that will be performed by software. The domain model is evaluated to ensure that the requirements are consistent, complete, and realistic, especially in the high risk areas. The model is evaluated to determine responses to error and fault conditions and to boundary and out-of-bounds conditions. As the domain engineering progresses into later phases, the requirements are traced forward. This will allow evaluation of the impact of changes to the domain artifacts.

Domain design V&V tasks focus on ensuring that the domain architecture satisfies the requirements expressed in the domain model. Each requirement in the domain model should trace to one or more items in the domain architecture (forward traceability), and each item in the domain architecture should trace back to one or more requirements in the domain model (reverse traceability). The domain architecture is evaluated to ensure that it is consistent, complete, and realistic. Interfaces between components are evaluated to ensure that the architecture supports the necessary communication between components in the architecture, users, and external systems. Planning and design of component testing are performed during this phase. The component testing should include error and fault scenarios, functional testing of critical activities, and response to boundary and out-of-bounds conditions.

Domain Implementation V&V tasks ensure that the domain components satisfy the requirements of the domain architecture and will satisfy the original user requirements. The components should have a forward and reverse tracing with the domain architecture. Components that are involved with performing critical actions should receive careful consideration. The interface implementation, both within components of the architecture and with systems outside the architecture, is evaluated to ensure that it meets the requirements of the domain architecture. Component test cases and test procedures are generated, and component testing is performed.

Integration test activities are explicitly omitted from the domain-level tasking, since integration testing is oriented toward application-specific testing. Some form of integration testing might be appropriate within domain-level V&V in the case where the architecture calls for specific domain components to be integrated in multiple systems. This limited form of integration testing could be done along with the component testing activities.

5.2 Correspondence Tasks

Correspondence analysis is a term not found in IEEE STD 1012. The term is used within this paper to describe the activities that are performed to provide assurance that an application artifact corresponds to a domain artifact; i.e., the application artifact is a correct implementation of the domain artifact. Four activities are to be performed during correspondence analysis:

- Map the application artifact to the corresponding domain artifact.
- Ensure that the application artifact has not been modified from the domain artifact without proper documentation.
- Ensure that the application artifact is a correct instantiation of the domain artifact.
- Obtain information on testing and analysis on a domain artifact to aid in V&V planning for the application artifact.

Correspondence analysis is performed between the corresponding phases of the domain engineering and application engineering life-cycles. The system specification for any system within the domain should correspond to the domain model. The system specification could involve instantiating, parameterizing, or simply satisfying the requirements expressed in the domain model. Any system-unique requirements should be explicit, and the rationale for not addressing these system-unique requirements within the domain model should be stated. Although some degree of correspondence analysis should be at least implicitly performed for all systems developed in accordance with the domain architecture, more care should be taken for systems with critical functions and for their critical areas of software.

The system architecture is analyzed to ensure that it satisfies the requirements specified in the domain architecture. Any variations should be documented along with the reason for the variation. The rationale for parameters chosen or options selected in constructing the system architecture from the domain architecture should be recorded.

The system components are analyzed to ensure correspondence to domain components. Again, variations, parameters, and options should be recorded along with their rationale. Baseline testing might be appropriate in order to compare variants of a domain component.

6. COMMUNICATING RESULTS

Communicating V&V work products and results is vital to avoiding the repetition of V&V tasks and to ensuring that potential reusers can properly assess the status of reusable components. V&V work products and results should be associated with the component and made available to domain and application engineers. In some cases, V&V efforts might be directed at a grouping of components rather than at an individual component, and this information should also be available. Groupings might include components that are expected to occur together in several applications, or might include variants of one domain artifact.

The information on similar components within the domain should be consistent in content and format, in order to allow the information to be easily used by both domain engineers and application engineers. The information that should be communicated include the following:

- V&V Planning Decisions and Rationale
- V&V Analysis Activities
- V&V Test Cases and Procedures
- V&V Results and Findings

7. V&V OF DOMAIN ARTIFACTS

This paper focuses on the issue of V&V within domain engineering, in the situation where the final systems would be subject to V&V even if the systems were not developed within a reuse environment. Many of the same justifications for performing V&V in a product line that includes critical systems also apply to V&V of general purpose reusable components. These general purpose components include domain artifacts for systems that are not critical, as well as reusable components that are developed for general usage rather than for a specific product line.

The Component Verification, Validation and Certification Working Group at WISR 8 found four considerations that should be used in determining the level of V&V of reusable components [Edwards and Wiede 1997]:

- Span of application – the number of components or systems that depend on the component
- Criticality – potential impact due to a fault in the component
- Marketability – degree to which a component would be more likely to be reused by a third party
- Lifetime – length of time that a component will be used

The domain architecture serves as the context for evaluating software components in a product-line environment. However, this architecture may not exist for general use components. The Working Group determined that the concept of validation was different for a general use component than for a component developed for a specific system or product line. In the latter case, validation refers to ensuring that the component meets the needs of the customer. A general use component has not one customer, but many customers, who are software developers rather then end-users. Hence validation of a general use component should involve the assurance (and supporting documentation) that the component satisfies a wide range of alternative usages, rather than the specific needs of a particular end-user.

8. RELATED WORK

Although work is lacking specifically in the area of V&V as applied to reuse-based software engineering, there is related work that is applicable to some of the tasks within the framework. Component certification was discussed in a previous section, and this work is certainly applicable (although not sufficient) for V&V activity at the domain level. The analysis of architectures is the focus of attention and discussion [Tracz 1996, Garlan 1995], but there is not as yet consensus on methods and approaches. One of the approaches being researched is a scenario-based analysis approach, Software Architecture Analysis Method [Kazman et al. 1995]. In the area of correspondence tasks, the Centre for Requirements and Foundations at Oxford is developing a tool (TOOR) to support tracing dependencies among evolving objects [Goguen 1996].

9. CONCLUSION

The framework for performing V&V in traditional application system development can be extended to reuse-based software engineering. The extended framework allows the V&V effort to be amortized over the systems within the domain or product line. Just as with V&V in application system development, V&V should be performed as part of an overall risk mitigation strategy within the domain or product line.

The primary motivation for V&V within domain engineering is to find and correct errors in the domain artifact in order to prevent the errors from being propagated to the application systems. This motivation is especially strong where the application systems perform critical functions. Even if there are no critical functions performed by the systems within the domain, V&V might be appropriate for a component that has the potential to be used in a large number of application systems.

ACKNOWLEDGEMENT

This work is funded by NASA Cooperative Agreement NCC 2-979 at the NASA/Ames IV&V Facility in Fairmont, WV.

REFERENCES

Addy, Edward A. (1996), "V&V Within Reuse-Based Software Engineering", Proceedings for the Fifth Annual Workshop on Software Reuse Education and Training, Reuse '96, http://www.asset.com/WSRD/conferences/proceedings/results/addy/addy.html.

Armitage, James W. (1993), "Process Guide for the DSSA Process Life Cycle", DSSA-PG-001, Software Engineering Institute Software Process Definition Project, Pittsburgh, PA

Duke, Eugene, L. (1989), "V&V of Flight and Mission-Critical Software", IEEE Software, 6, 3, 39-45.

Dunn, Michael F. and John C. Knight (1993), "Certification of Reusable Software Parts," Technical Report CS-93-41, University of Virginia, Charlottesville, VA.

Edwards, Stephen H. and Bruce W. Wiede (1997), "WISR8: 8th Annual Workshop on SW Reuse", Software Engineering Notes, 22, 5, 17-32.

Garlan, David (1995), "First International Workshop on Architectures for Software Systems Workshop Summary", Software Engineering Notes, 20, 3, 84-89.

Goguen, Joseph A. (1996), "Parameterized Programming and Software Architecture," In Proceedings of the Fourth International Conference on Software Reuse, IEEE Computer Society Press, Los Alamitos, CA, pp. 2-10.

Katz, Susan, Christopher Dabrowski, Kathryn Miles and Margaret Law (1994), "Glossary of Software Reuse Terms," NIST Special Publication 500-222, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD.

IEEE STD 610.12-1990, IEEE Standard Glossary of Software Engineering Technology, Institute of Electrical and Electronics Engineers, Inc., New York, NY.

IEEE STD 1012-1986 (R 1992), IEEE Standard for Software Verification and Validation Plans, Institute of Electrical and Electronics Engineers, Inc., New York, NY.

IEEE STD 1059-1993, IEEE Guide for Software Verification and Validation Plans, Institute of Electrical and Electronics, Inc., New York, NY.

Kazman, Rick, Gregory Abowd, Len Bass, and Paul Clements, "Scenario-Based Analysis of Software Architecture," IEEE Software, 13, 6, 47-55.

Lewis, Robert O. (1992), Independent Verification and Validation, A Life Cycle Engineering Process for Quality Software, John Wiley & Sons, New York, NY.

Makowsky, Lawrence C. (1992), "A Guide to Independent Verification and Validation of Computer Software," USA-BRDEC-TR//2516, United States Army Belvoir Research, Development and Engineering Center, Fort Belvoir, VA.

Poore, J.H., Theresa Pepin, Murali Sitaraman, and Frances L. Van Scoy (1992), "Criteria and Implementation Procedures for Evaluating Reusable Software Engineering Assets," DTIC AD-B166803, prepared for IBM Corportation Federal Sectors Division, Gaithersburg, MD.

Software Productivity Solutions, Inc. (1996), "Certification of Reusable Software Components, Volume 2 – Certification Framework," prepared for Rome Laboratory/C3CB, Griffiss AFB, NY.

Tracz, Will (1996), "Test and Analysis of Software Architectures," In Proceedings of the Internatinal Symposium on Software Testing and Analysis (ISSTA '96), ACM Press, New York, NY, pp 1-3.

Unisys, Valley Forge Engineering Center, and EWA, Inc. (1994), "Component Provider's and Tool Developer's Handbook," STARS-VC-B017/001/00, prepared for Electronic Systems Center, Air Force Material Command, USAF, Hanscom AFB, MA.

Wallace, Dolores R. and Roger U. Fujii (1989a), Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards," NIST Special Publication 500-165, National Institute of Standards and Technology, Gaithersburg, MD.

Wallace, Dolores R. and Roger U. Fujii (1989b), "Software Verification and Validation: An Overview", IEEE Software, 6, 3, 10-17.